IT - 314

SOFTWARE ENGINEERING

Lab 7 - Program Inspection, Debugging and Static Analysis

Group No - 11

Smit Godhani - 202201162

Program Inspection and Code Debugging

1. ArmStrong

A. Program Inspection

1. The program incorrectly extracts the remainder using num / 10 instead of num % 10. Also, it should reduce num using num = num / 10 in the loop, not num = num % 10.
2. Data Reference Errors (wrong remainder extraction) and Computation Errors (incorrect loop update).
3. The program doesn't check if the input is valid like negative numbers or non-numeric input.
4. Yes, it helps find mistakes in logic and math that affect the program's output.

B. Code Debugging

1. The program incorrectly extracts the remainder using num / 10 instead of num % 10. It should reduce num using num = num / 10 in the loop.
2. Two breakpoints are needed to fix those errors.
   - Change remainder = num / 10; to remainder = num % 10;
   - Change num = num % 10; to num = num / 10;
3. Complete Executable Code:

```
class Armstrong{
    public static void main(String args[]){
        int num = Integer.parseInt(args[0]);
        int n = num;
        int check = 0, remainder;
        while(num > 0){
            remainder = num % 10;
            check = check + (int)Math.pow(remainder,3);
            num = num / 10;
        }
        if(check == n)
            System.out.println(n + " is an Armstrong Number");
        else
            System.out.println(n + " is not an Armstrong Number");
    }
}
```

2. GCD and LCM

A. Program Inspection

1. There are two errors.
   - In the gcd method, the loop condition should be while(a % b != 0) instead of while(a % b == 0).
   - In the lcm method, the condition should be if(a % x == 0 && a % y == 0) instead of if(a % x != 0 && a % y != 0).
2. Logic Errors and Computation Errors are most effective as they help in identifying mistakes in flow and calculations.
3. Input validation errors, such as negative numbers or non-numeric input, are not identified.
4. Yes, it helps catch logical and mathematical errors that affect program output.

B. Code Debugging

1. There are two errors.
   - In the gcd method, the loop condition should be changed from while(a % b == 0) to while(a % b != 0), which prevents the loop from executing correctly.
   - In the lcm method, the condition should be changed from if(a % x != 0 && a % y != 0) to if(a % x == 0 && a % y == 0), which incorrectly identifies the least common multiple.
2. Two breakpoints are needed to fix those errors.
   - Set a breakpoint in the gcd method and change the loop condition to while(a % b != 0).
   - Set a breakpoint in the lcm method and change the condition to if(a % x == 0 && a % y == 0).
3. Complete Executable Code:

```
import java.util.Scanner;

public class GCD_LCM {
    static int gcd(int x, int y) {
        int r = 0, a, b;
        a = (x > y) ? y : x;
        b = (x < y) ? x : y;

        r = b;
        while (a % b != 0) {
            r = a % b;
            a = b;
            b = r;
        }
        return r;
    }

    static int lcm(int x, int y) {
        int a;
        a = (x > y) ? x : y;
        while (true) {
            if (a % x == 0 && a % y == 0)
                return a;
            ++a;
        }
    }

    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the two numbers: ");
        int x = input.nextInt();
        int y = input.nextInt();

        System.out.println("The GCD of two numbers is: " + gcd(x, y));
        System.out.println("The LCM of two numbers is: " + lcm(x, y));
        input.close();
    }
}
```

3. Knapsack

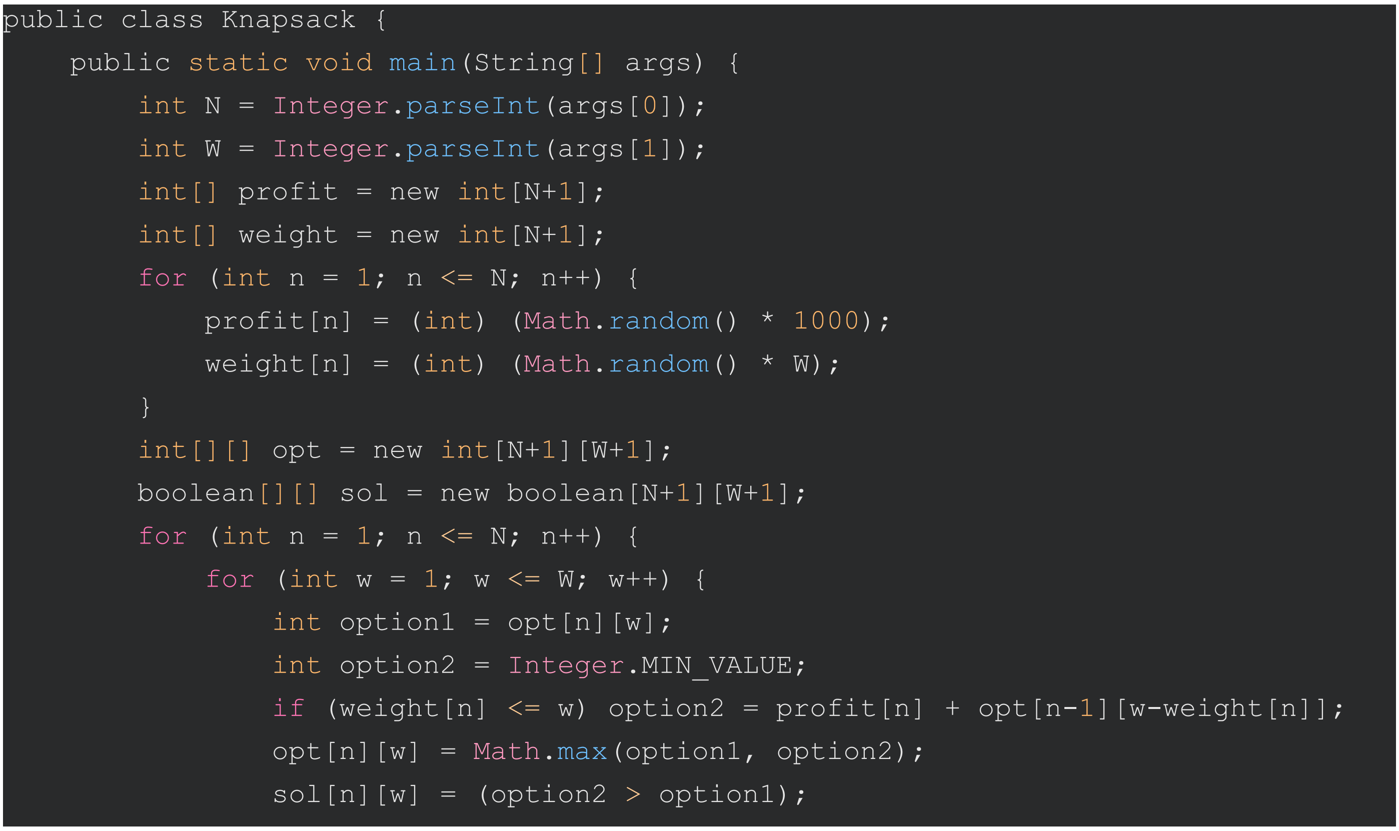
A. Program Inspection

1. There are three errors.
   - In the loop for `option1`, it uses `n++` instead of just `n` which causes an out-of-bounds error.
   - In the `option2` calculation, it uses `profit[n-2]` instead of `profit[n]` to correctly access the profit for item `n`.
   - The condition for taking an item is wrong; it should check `weight[n] <= w` instead of `weight[n] > w`.
2. Logic Errors and Array Index Errors are most effective as they help in identifying mistakes in flow and calculations.

3. The program does not handle cases where the inputs are invalid like negative numbers or non-numeric input.
5. Yes, it helps identify logical flaws and ensures the algorithm functions as intended.

B. Code Debugging

1. There are five errors.
    - In the `option1` calculation, it uses `n++` which causes an out-of-bounds error.
    - In the `option2` calculation, it incorrectly accesses profit with `profit[n-2]` instead of `profit[n]`.
    - The condition to check if an item can be taken uses `weight[n] > w` instead of `weight[n] <= w`.
2. Three breakpoints are needed to fix those errors.
    - Set a breakpoint at `int option1 = opt[n++][w];` and change it to `int option1 = opt[n][w];`.
    - Set a breakpoint at `option2 = profit[n-2] + opt[n-1][w-weight[n]];` and change it to `option2 = profit[n] + opt[n-1][w-weight[n]];`.
    - Set a breakpoint at the condition check and change it to `if (weight[n] <= w)`.
3. Complete Executable Code:



```
public class Knapsack {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int W = Integer.parseInt(args[1]);
        int[] profit = new int[N+1];
        int[] weight = new int[N+1];
        for (int n = 1; n <= N; n++) {
            profit[n] = (int) (Math.random() * 1000);
            weight[n] = (int) (Math.random() * W);
        }
        int[][] opt = new int[N+1][W+1];
        boolean[][] sol = new boolean[N+1][W+1];
        for (int n = 1; n <= N; n++) {
            for (int w = 1; w <= W; w++) {
                int option1 = opt[n][w];
                int option2 = Integer.MIN_VALUE;
                if (weight[n] <= w) option2 = profit[n] + opt[n-1][w-weight[n]];
                opt[n][w] = Math.max(option1, option2);
                sol[n][w] = (option2 > option1);
            }
        }
        boolean[] take = new boolean[N+1];
        for (int n = N, w = W; n > 0; n--) {
            if (sol[n][w]) { take[n] = true;  w = w - weight[n]; }
            else           { take[n] = false;                    }
        }
        System.out.println("item" + "\t" + "profit" + "\t" + "weight" + "\t" +
"take");
        for (int n = 1; n <= N; n++) {
            System.out.println(n + "\t" + profit[n] + "\t" + weight[n] + "\t" +
take[n]);
        }
    }
}
```

4. Magic Number

A. Program Inspection

1. There are three errors. The sum variable should be reset to zero before the inner loop. The inner loop should check `sum != 0` instead of `sum == 0`, and `s` should accumulate the digits using `s += (sum % 10)`.
2. Logic Errors are most effective as they help in identifying mistakes in flow and calculations.
3. The program does not handle invalid inputs like negative numbers or non-numeric input.
4. Yes, it helps identify logical flaws and ensures the algorithm functions as intended.

B. Code Debugging

1. There are three errors.
   - The inner loop should check (`sum != 0`).
   - `s` should accumulate digits using `s += (sum % 10)`.
   - `sum` should be reset to zero at the start of the loop.

2. Three breakpoints are needed to fix those errors.
   - Set a breakpoint at `while(sum == 0)` and change it to `while(sum != 0)`.

- Set a breakpoint at `s = s * (sum / 10);` and change it to `s += (sum % 10);`.
- Set a breakpoint before `sum = sum % 10;` to reset sum to zero before accumulating digits.

3. Complete Executable Code:

```
import java.util.*;
public class MagicNumberCheck {
    public static void main(String args[]) {
        Scanner ob = new Scanner(System.in);
        System.out.println("Enter the number to be checked.");
        int n = ob.nextInt();
        int sum = 0, num = n;
        while (num > 9) {
            sum = num; int s = 0;
            while (sum != 0) {
                s += (sum % 10);
                sum = sum / 10;
            }
            num = s;
        }
        if (num == 1) {
            System.out.println(n + " is a Magic Number.");
        } else {
            System.out.println(n + " is not a Magic Number.");
        }
    }
}
```

5. Merge Sort

A. Program Inspection

1. There are four errors.
   - The method leftHalf(array + 1) should be leftHalf(array), and rightHalf(array - 1) should be rightHalf(array).
   - The merge method should take array instead of (array + 1) and (array - 1) for merging.
   - The left++ and right-- are incorrect; it should be left and right.
2. Logic Errors are most effective as they help in identifying mistakes in the flow of the algorithm.

3. The program does not handle cases where the input array is null or has a length of zero.
4. Yes, it helps identify logical flaws in the implementation of the sorting algorithm.

B. Code Debugging

1. There are four errors. The calls to leftHalf and rightHalf should pass array, not array + 1 or array - 1. The merge method should use merge(array, left, right) without incrementing/decrementing left and right.
2. Four breakpoints are needed to fix those errors.
   - Set a breakpoint at int[] left = leftHalf(array + 1); and change it to int[] left = leftHalf(array);.
   - Set a breakpoint at int[] right = rightHalf(array - 1); and change it to int[] right = rightHalf(array);.
   - Set a breakpoint at merge(array, left++, right--); and change it to merge(array, left, right);.
   - Verify that the merge method is correctly merging arrays without any errors in indexing.
3. Complete Executable Code:

```
import java.util.*;

public class MergeSort {
    public static void main(String[] args) {
        int[] list = {14, 32, 67, 76, 23, 41, 58, 85};
        System.out.println("before: " + Arrays.toString(list));
        mergeSort(list);
        System.out.println("after:  " + Arrays.toString(list));
    }

    public static void mergeSort(int[] array) {
        if (array.length > 1) {
            int[] left = leftHalf(array);
            int[] right = rightHalf(array);
            mergeSort(left);
            mergeSort(right);
            merge(array, left, right);
        }
    }

    public static int[] leftHalf(int[] array) {
        int size1 = array.length / 2;
        int[] left = new int[size1];
        for (int i = 0; i < size1; i++) {
            left[i] = array[i];
        }
        return left;
    }

    public static int[] rightHalf(int[] array) {
        int size1 = array.length / 2;
        int size2 = array.length - size1;
        int[] right = new int[size2];
        for (int i = 0; i < size2; i++) {
            right[i] = array[i + size1];
        }
        return right;
    }

    public static void merge(int[] result, int[] left, int[] right) {
        int i1 = 0;
        int i2 = 0;
        for (int i = 0; i < result.length; i++) {
            if (i2 >= right.length || (i1 < left.length && left[i1] <= right[i2]))
{
                result[i] = left[i1];
                i1++;
            } else {
                result[i] = right[i2];
                i2++;
            }
        }
    }
}
```

6. Multiply Matrics

A. Program Inspection

1. There are Three errors.
   - In the multiplication loop, the indices are incorrect: first[c-1][c-k] should be first[c][k], and second[k-1][k-d] should be second[k][d].
   - The second matrix input prompt incorrectly states "Enter the number of rows and columns of first matrix" instead of "Enter the number of rows and columns of second matrix."

2. Logic Errors are most effective for identifying mistakes in the flow of matrix multiplication.
3. The program does not check for non-integer inputs when reading matrix elements.
4. Yes, it helps identify logical flaws and ensures the matrix multiplication works correctly.

B. Code Debugging

1. There are three errors. The multiplication loop uses incorrect indices for accessing matrix elements.
2. Three breakpoints are needed to fix those errors.
   - Set a breakpoint at sum = sum + first[c-1][c-k]*second[k-1][k-d]; and change it to sum = sum + first[c][k] * second[k][d];.
   - Set a breakpoint to correct the input prompt from "first matrix" to "second matrix" when reading dimensions.
   - Ensure the loop and indexing logic correctly reflect the intended multiplication process.
3. Complete Executable Code:

```
import java.util.Scanner;

class MatrixMultiplication {
    public static void main(String args[]) {
        int m, n, p, q, sum = 0, c, d, k;

        Scanner in = new Scanner(System.in);
        System.out.println("Enter the number of rows and columns of first
matrix");
        m = in.nextInt();
        n = in.nextInt();

        int first[][] = new int[m][n];

        System.out.println("Enter the elements of first matrix");
        for (c = 0; c < m; c++)
            for (d = 0; d < n; d++)
                first[c][d] = in.nextInt();

        System.out.println("Enter the number of rows and columns of second
matrix");
        p = in.nextInt();
        q = in.nextInt();

        if (n != p)
```

```
            System.out.println("Matrices with entered orders can't be multiplied
with each other.");
        else {
            int second[][] = new int[p][q];
            int multiply[][] = new int[m][q];

            System.out.println("Enter the elements of second matrix");
            for (c = 0; c < p; c++)
                for (d = 0; d < q; d++)
                    second[c][d] = in.nextInt();

            for (c = 0; c < m; c++) {
                for (d = 0; d < q; d++) {
                    for (k = 0; k < n; k++) {
                        sum = sum + first[c][k] * second[k][d];
                    }
                    multiply[c][d] = sum;
                    sum = 0;
                }
            }

            System.out.println("Product of entered matrices:-");
            for (c = 0; c < m; c++) {
                for (d = 0; d < q; d++)
                    System.out.print(multiply[c][d] + "\t");
                System.out.print("\n");
            }
        }
    }
}
```

7. Quadratic Probing

A. Program Inspection

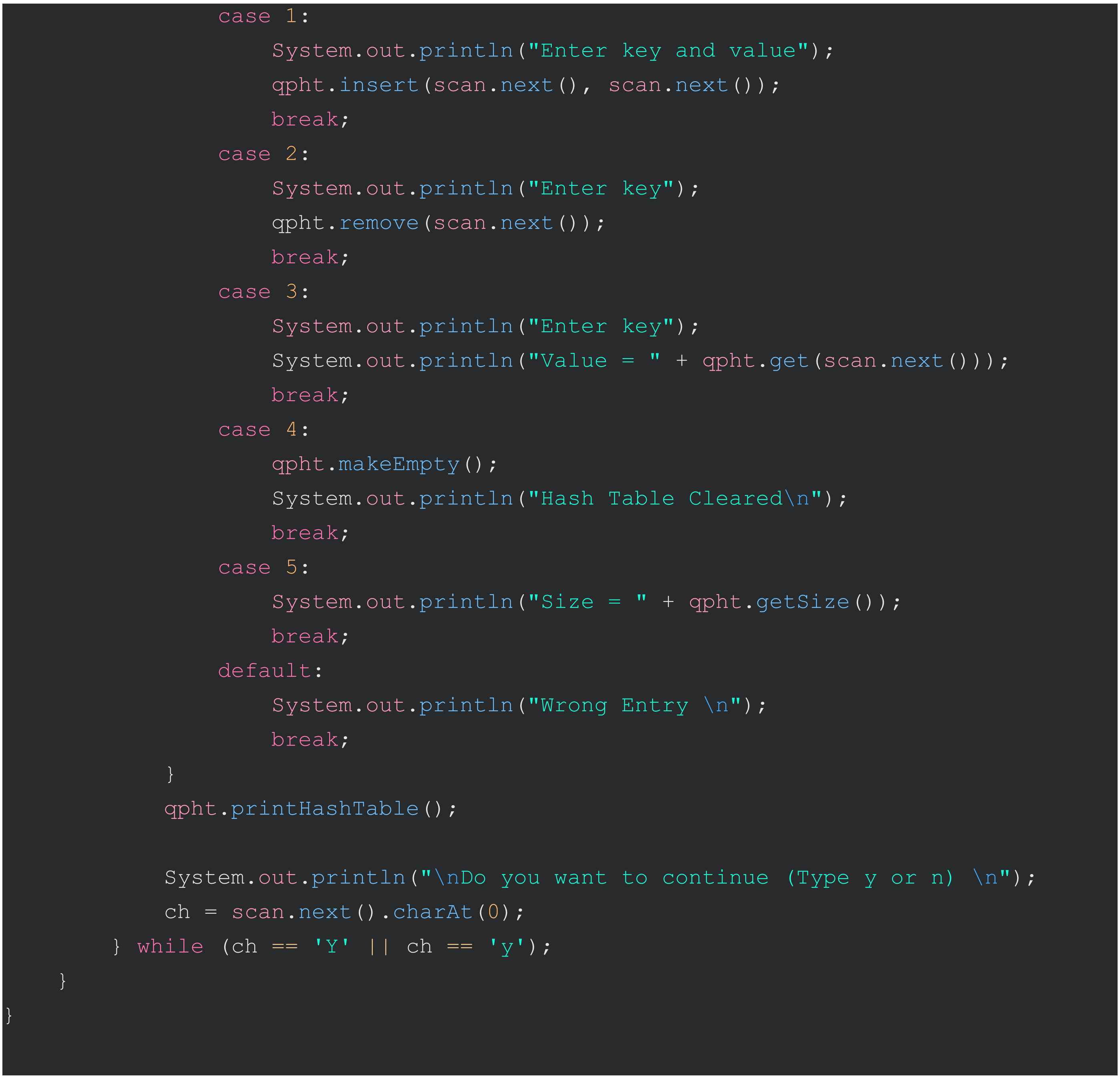
1. There are Two errors.
   - In the insert method, the expression i + = (i + h / h--) % maxSize; has an extra space and incorrect logic. It should be i = (i + h * h) % maxSize; and should increment h properly.
   - In the get method, the expression i = (i + h * h++) % maxSize; also has incorrect logic. It should be i = (i + h * h) % maxSize; with correct increment of h.

2. Logic Errors are effective for catching issues in the hashing and probing logic.
3. The program does not handle the scenario where the hash table is full during insertion.
4. Yes, logic error inspection helps to identify flaws in hashing and insertion mechanics.

B. Code Debugging

1. There are two errors.
    - Incorrect logic in the insert and get methods for updating i and h.
    - Missing handling for a full hash table during insertion.
2. Set breakpoints at the logic statements inside the insert and get methods to check values of i and h.
3. Complete Executable Code:

```
import java.util.Scanner;

class QuadraticProbingHashTable {
    private int currentSize, maxSize;
    private String[] keys;
    private String[] vals;

    public QuadraticProbingHashTable(int capacity) {
        currentSize = 0;
        maxSize = capacity;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }

    public void makeEmpty() {
        currentSize = 0;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }

    public int getSize() {
        return currentSize;
    }

    public boolean isFull() {
        return currentSize == maxSize;
    }

    public boolean isEmpty() {
        return getSize() == 0;
    }

    public boolean contains(String key) {
        return get(key) != null;
    }

    private int hash(String key) {
        return Math.abs(key.hashCode()) % maxSize;
    }

    public void insert(String key, String val) {
        if (isFull()) {
            System.out.println("Hash table is full.");
            return;
        }

        int tmp = hash(key);
        int i = tmp, h = 1;

        do {
            if (keys[i] == null) {
                keys[i] = key;
                vals[i] = val;
                currentSize++;
                return;
            }
            if (keys[i].equals(key)) {
                vals[i] = val;
                return;
            }
            i = (tmp + h * h) % maxSize;
            h++;
        } while (i != tmp);
    }

    public String get(String key) {
        int i = hash(key), h = 1;
        while (keys[i] != null) {
            if (keys[i].equals(key))
                return vals[i];
            i = (i + h * h) % maxSize;
            h++;
        }
        return null;
    }

    public void remove(String key) {
        if (!contains(key))
            return;

        int i = hash(key), h = 1;
        while (!key.equals(keys[i]))
            i = (i + h * h) % maxSize;

        keys[i] = vals[i] = null;

        for (i = (i + h * h) % maxSize; keys[i] != null; i = (i + h * h) %
maxSize) {
            String tmp1 = keys[i], tmp2 = vals[i];
            keys[i] = vals[i] = null;
            currentSize--;
            insert(tmp1, tmp2);
        }
        currentSize--;
    }

    public void printHashTable() {
        System.out.println("\nHash Table: ");
        for (int i = 0; i < maxSize; i++)
            if (keys[i] != null)
                System.out.println(keys[i] + " " + vals[i]);
        System.out.println();
    }
}

public class QuadraticProbingHashTableTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Hash Table Test\n\n");
        System.out.println("Enter size");
        QuadraticProbingHashTable qpht = new
QuadraticProbingHashTable(scan.nextInt());

        char ch;
        do {
            System.out.println("\nHash Table Operations\n");
            System.out.println("1. insert ");
            System.out.println("2. remove");
            System.out.println("3. get");
            System.out.println("4. clear");
            System.out.println("5. size");

            int choice = scan.nextInt();
            switch (choice) {
```



```
                case 1:
                    System.out.println("Enter key and value");
                    qpht.insert(scan.next(), scan.next());
                    break;
                case 2:
                    System.out.println("Enter key");
                    qpht.remove(scan.next());
                    break;
                case 3:
                    System.out.println("Enter key");
                    System.out.println("Value = " + qpht.get(scan.next()));
                    break;
                case 4:
                    qpht.makeEmpty();
                    System.out.println("Hash Table Cleared\n");
                    break;
                case 5:
                    System.out.println("Size = " + qpht.getSize());
                    break;
                default:
                    System.out.println("Wrong Entry \n");
                    break;
            }
            qpht.printHashTable();

            System.out.println("\nDo you want to continue (Type y or n) \n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y' || ch == 'y');
    }
}
```

8. Sorting Array

A. Program Inspection

1. There are four errors.
    - The class name Ascending _Order contains an invalid space; it should be AscendingOrder.
    - The outer loop condition in the sorting logic is incorrect. It should be for (int i = 0; i < n; i++) instead of for (int i = 0; i >= n; i++);.
    - There is a semicolon at the end of the outer loop, causing the inner loop to run incorrectly.

    - The sorting condition should be if (a[i] > a[j]) to sort in ascending order properly.
2. Logic Errors are most effective for identifying mistakes in the flow of matrix multiplication.
3. The program does not handle non-integer inputs when reading elements.
4. Yes, it helps identify logical flaws and ensures that the sorting works correctly.

B. Code Debugging

1. There are four errors.
    - The class name Ascending _Order contains an invalid space; it should be AscendingOrder.
    - The outer loop condition in the sorting logic is incorrect. It should be for (int i = 0; i < n; i++) instead of for (int i = 0; i >= n; i++);.
    - There is a semicolon at the end of the outer loop, causing the inner loop to run incorrectly.
    - The sorting condition should be if (a[i] > a[j]) to sort in ascending order properly.
2. Three breakpoints are needed to fix those errors.
    - Set a breakpoint at the class declaration and correct the name to AscendingOrder.
    - Set a breakpoint at for (int i = 0; i >= n; i++); and change the condition to for (int i = 0; i < n; i++).
    - Remove the semicolon after the outer loop to ensure the inner loop executes correctly.
3. Complete Executable Code:

```
import java.util.Scanner;

public class AscendingOrder {
    public static void main(String[] args) {
        int n, temp;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter no. of elements you want in array:");
        n = s.nextInt();
        int a[] = new int[n];
        System.out.println("Enter all the elements:");
        for (int i = 0; i < n; i++) {
            a[i] = s.nextInt();
        }
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) {
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }
        System.out.print("Ascending Order:");
        for (int i = 0; i < n - 1; i++) {
            System.out.print(a[i] + ",");
        }
        System.out.print(a[n - 1]);
    }
}
```

9. Stack Implementation

A. Program Inspection

1. There are three errors.
    - In the push method, the top index is decremented before assigning a value, which causes incorrect behavior.
    - The pop method does not return the popped value, which makes it less useful.
    - In the display method, the loop condition is incorrect; it should be i <= top instead of i > top.
2. Logic Errors are most effective for identifying mistakes in stack operations and control flow.
3. The program does not handle non-integer inputs when pushing values onto the stack.
4. Yes, it helps identify logical flaws and ensures that the stack operations work correctly.

B. Code Debugging

1. There are three errors.
    - The push method incorrectly decrements top before adding a value.

   - The pop method needs to return the popped value for usability.
   - The display method has the wrong loop condition.
2. Three breakpoints are needed to fix those errors.
   - Set a breakpoint at the top--; line in the push method and change it to top++;.
   - Set a breakpoint in the pop method to ensure it returns the popped value.
   - Set a breakpoint at for(int i=0;i>top;i++) in the display method and change it to for(int i=0;i<=top;i++).
3. Complete Executable Code:

```
import java.util.Arrays;

public class StackMethods {
    private int top;
    int size;
    int[] stack;

    public StackMethods(int arraySize) {
        size = arraySize;
        stack = new int[size];
        top = -1;
    }

    public void push(int value) {
        if (top == size - 1) {
            System.out.println("Stack is full, can't push a value");
        } else {
            top++;
            stack[top] = value;
        }
    }

    public int pop() {
        if (!isEmpty()) {
            return stack[top--];
        } else {
            System.out.println("Can't pop...stack is empty");
            return -1; // Indicate stack is empty
        }
    }

    public boolean isEmpty() {
        return top == -1;
    }
    public void display() {
        for (int i = 0; i <= top; i++) {
            System.out.print(stack[i] + " ");
        }
        System.out.println();
    }
}

public class StackReviseDemo {
    public static void main(String[] args) {
        StackMethods newStack = new StackMethods(5);
        newStack.push(10);
        newStack.push(1);
        newStack.push(50);
        newStack.push(20);
        newStack.push(90);

        newStack.display();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.pop();
        newStack.display();
    }
}
```

10. Tower Of Hanoi

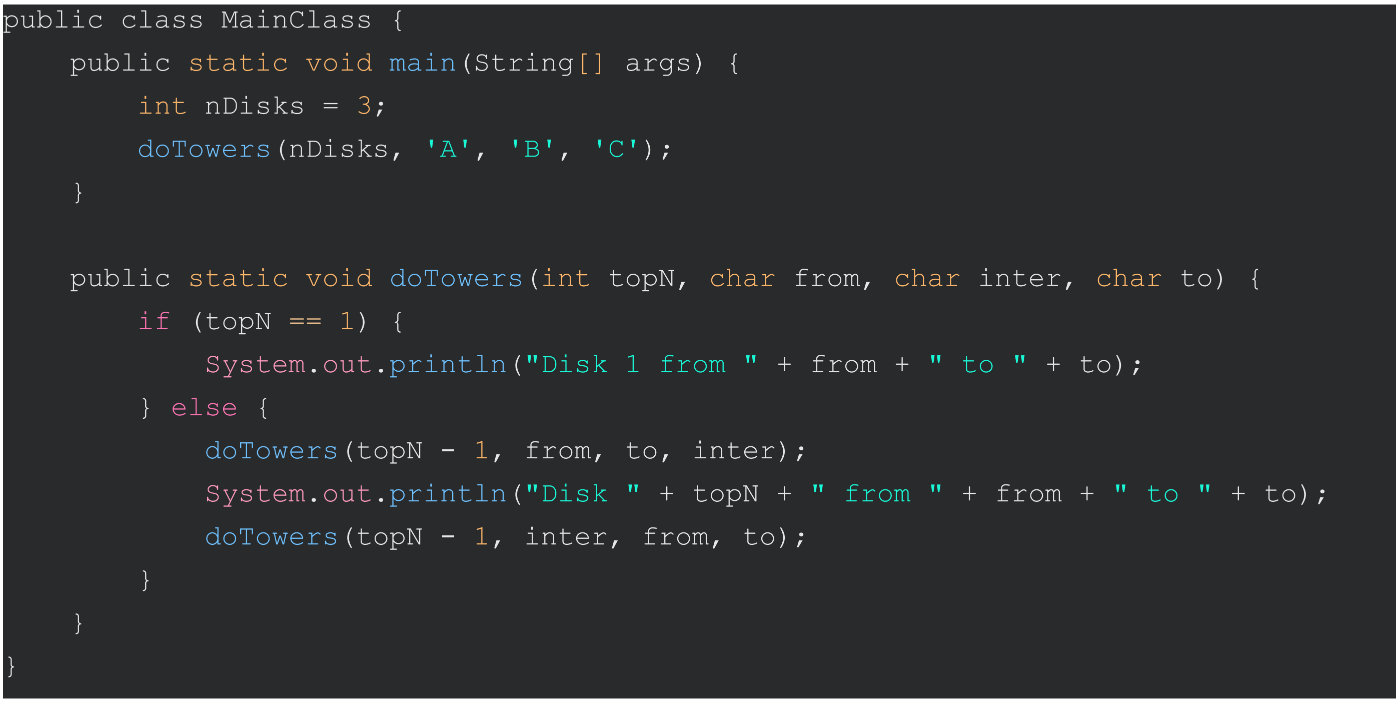
A. Program Inspection

1. There are three errors.
    - The recursive call doTowers(topN ++, inter--, from+1, to+1) contains incorrect syntax and logic. The ++ and -- operators do not work as intended for the parameters.
    - The parameters from and to are being incremented inappropriately, leading to incorrect character values.
    - The base case does not handle the movement of more than one disk correctly.
2. Logic Errors are most effective for identifying mistakes in recursive function calls and control flow.

3. The program does not check for negative or zero disk values, which could lead to unexpected behavior.
4. Yes, it helps identify logical flaws and ensures that the Tower of Hanoi logic is implemented correctly.

B. Code Debugging

1. There are three errors.
    - The recursive call uses incorrect syntax (topN ++ and inter--).
    - The parameters from and to are incorrectly manipulated.
    - The base case and recursive case do not correctly manage the disk movements.
2. Three breakpoints are needed to fix those errors.
    - Set a breakpoint at doTowers(topN ++, inter--, from+1, to+1) and change it to doTowers(topN - 1, inter, from, to).
    - Ensure that the parameters from, inter, and to are passed without modifications.
    - Adjust the recursive calls to ensure proper movement of disks.
3. Complete Executable Code:



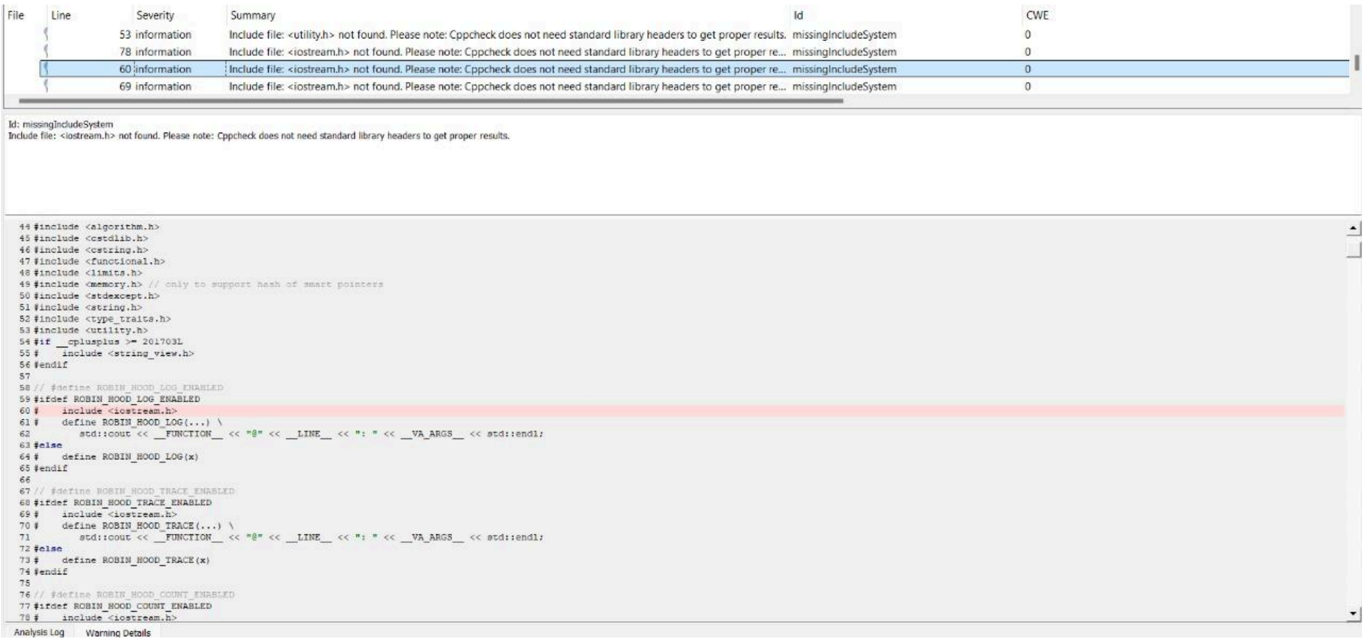
```java
public class MainClass {
    public static void main(String[] args) {
        int nDisks = 3;
        doTowers(nDisks, 'A', 'B', 'C');
    }

    public static void doTowers(int topN, char from, char inter, char to) {
        if (topN == 1) {
            System.out.println("Disk 1 from " + from + " to " + to);
        } else {
            doTowers(topN - 1, from, to, inter);
            System.out.println("Disk " + topN + " from " + from + " to " + to);
            doTowers(topN - 1, inter, from, to);
        }
    }
}
```

Static Analysis Tools

Github Code Link:

robin-hood-hashing/src/include/robin_hood.h at master · martinus/robin-hood-hashing · GitHub

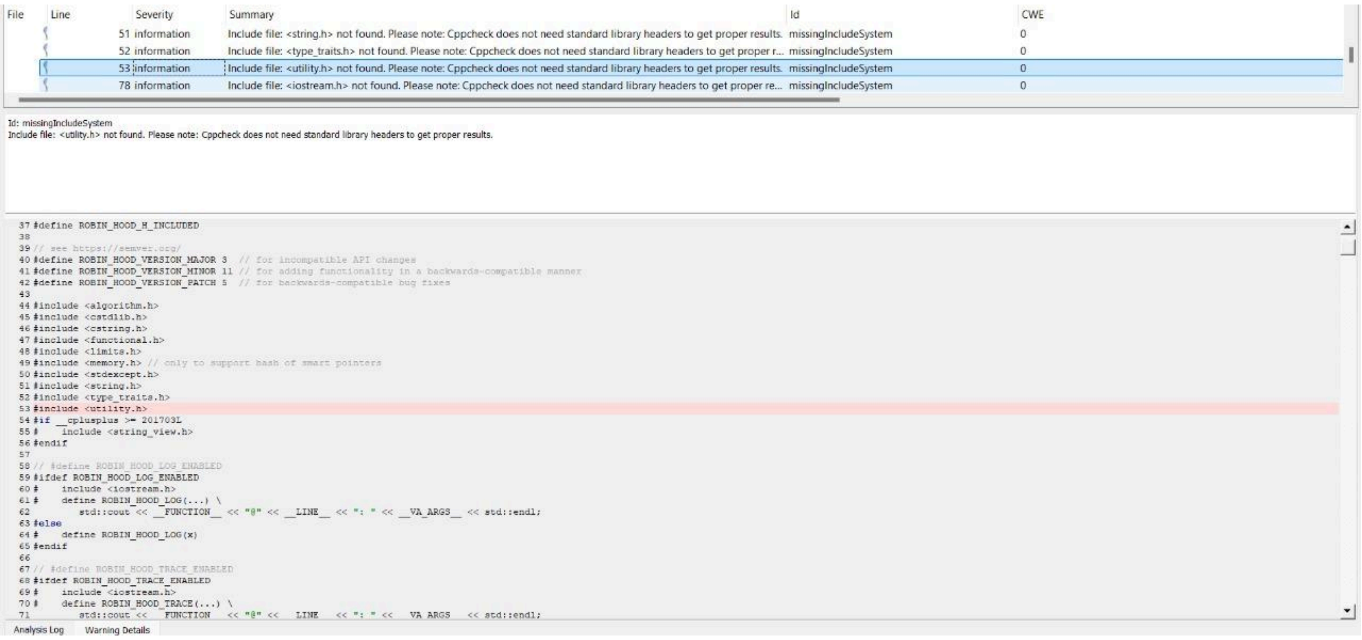
JPG Results:



File	Line	Severity	Summary	Id	CWE
	53	information	Include file: <utility.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem	0
	78	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0
	60	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0
	69	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0

Id: missingIncludeSystem
Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

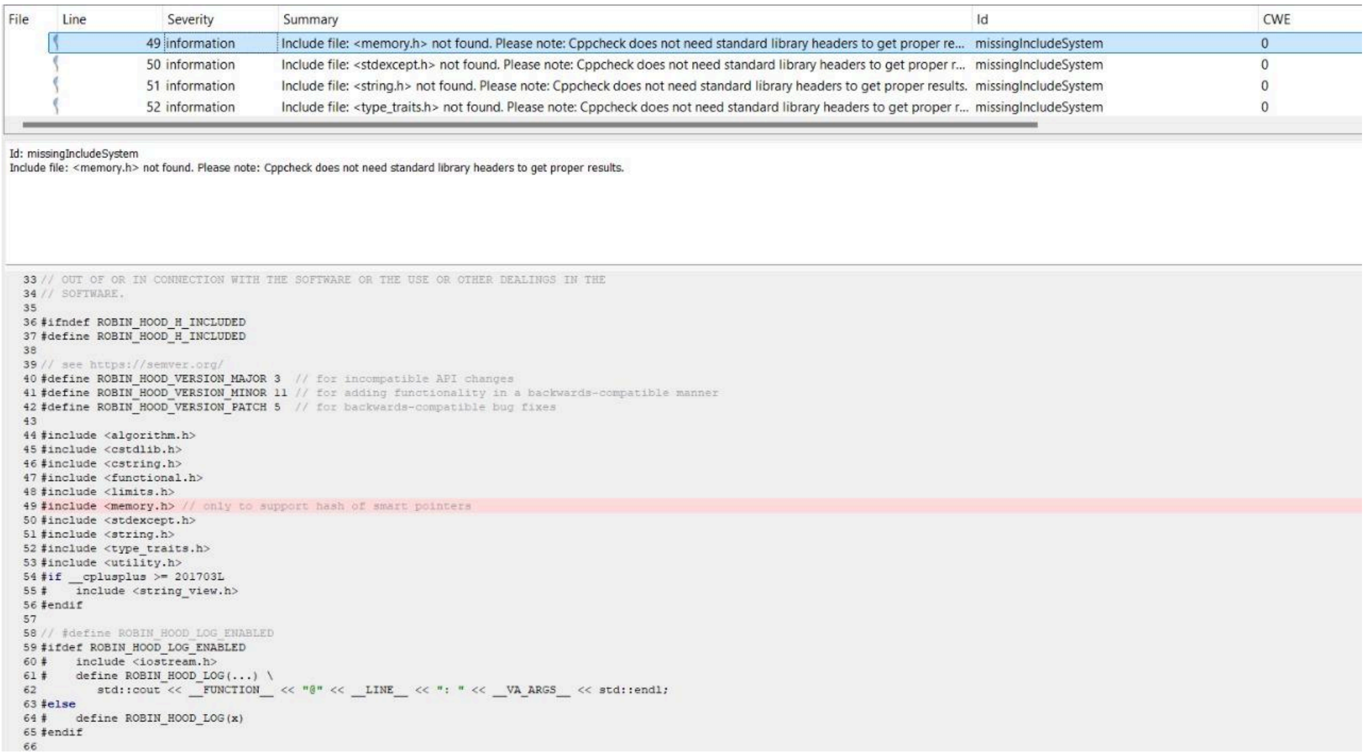
```
44 #include <algorithm.h>
45 #include <cstdlib.h>
46 #include <cstring.h>
47 #include <functional.h>
48 #include <limits.h>
49 #include <memory.h> // only to support hash of smart pointers
50 #include <stdexcept.h>
51 #include <string.h>
52 #include <type_traits.h>
53 #include <utility.h>
54 #if __cplusplus >= 201703L
55 #    include <string_view.h>
56 #endif
57
58 // #define ROBIN_HOOD_LOG_ENABLED
59 #ifdef ROBIN_HOOD_LOG_ENABLED
60 #    include <iostream.h>
61 #    define ROBIN_HOOD_LOG(...) \
62         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
63 #else
64 #    define ROBIN_HOOD_LOG(x)
65 #endif
66
67 // #define ROBIN_HOOD_TRACE_ENABLED
68 #ifdef ROBIN_HOOD_TRACE_ENABLED
69 #    include <iostream.h>
70 #    define ROBIN_HOOD_TRACE(...) \
71         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
72 #else
73 #    define ROBIN_HOOD_TRACE(x)
74 #endif
75
76 // #define ROBIN_HOOD_COUNT_ENABLED
77 #ifdef ROBIN_HOOD_COUNT_ENABLED
78 #    include <iostream.h>
```



File	Line	Severity	Summary	Id	CWE
	51	information	Include file: <string.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem	0
	52	information	Include file: <type_traits.h> not found. Please note: Cppcheck does not need standard library headers to get proper r...	missingIncludeSystem	0
	53	information	Include file: <utility.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem	0
	78	information	Include file: <iostream.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0

Id: missingIncludeSystem
Include file: <utility.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

```
37 #define ROBIN_HOOD_H_INCLUDED
38
39 // see https://semver.org/
40 #define ROBIN_HOOD_VERSION_MAJOR 3  // for incompatible API changes
41 #define ROBIN_HOOD_VERSION_MINOR 11 // for adding functionality in a backwards-compatible manner
42 #define ROBIN_HOOD_VERSION_PATCH 5  // for backwards-compatible bug fixes
43
44 #include <algorithm.h>
45 #include <cstdlib.h>
46 #include <cstring.h>
47 #include <functional.h>
48 #include <limits.h>
49 #include <memory.h> // only to support hash of smart pointers
50 #include <stdexcept.h>
51 #include <string.h>
52 #include <type_traits.h>
53 #include <utility.h>
54 #if __cplusplus >= 201703L
55 #    include <string_view.h>
56 #endif
57
58 // #define ROBIN_HOOD_LOG_ENABLED
59 #ifdef ROBIN_HOOD_LOG_ENABLED
60 #    include <iostream.h>
61 #    define ROBIN_HOOD_LOG(...) \
62         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
63 #else
64 #    define ROBIN_HOOD_LOG(x)
65 #endif
66
67 // #define ROBIN_HOOD_TRACE_ENABLED
68 #ifdef ROBIN_HOOD_TRACE_ENABLED
69 #    include <iostream.h>
70 #    define ROBIN_HOOD_TRACE(...) \
71         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
```



File	Line	Severity	Summary	Id	CWE
	49	information	Include file: <memory.h> not found. Please note: Cppcheck does not need standard library headers to get proper re...	missingIncludeSystem	0
	50	information	Include file: <stdexcept.h> not found. Please note: Cppcheck does not need standard library headers to get proper r...	missingIncludeSystem	0
	51	information	Include file: <string.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.	missingIncludeSystem	0
	52	information	Include file: <type_traits.h> not found. Please note: Cppcheck does not need standard library headers to get proper r...	missingIncludeSystem	0

Id: missingIncludeSystem
Include file: <memory.h> not found. Please note: Cppcheck does not need standard library headers to get proper results.

```
33 // OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
34 // SOFTWARE.
35
36 #ifndef ROBIN_HOOD_H_INCLUDED
37 #define ROBIN_HOOD_H_INCLUDED
38
39 // see https://semver.org/
40 #define ROBIN_HOOD_VERSION_MAJOR 3  // for incompatible API changes
41 #define ROBIN_HOOD_VERSION_MINOR 11 // for adding functionality in a backwards-compatible manner
42 #define ROBIN_HOOD_VERSION_PATCH 5  // for backwards-compatible bug fixes
43
44 #include <algorithm.h>
45 #include <cstdlib.h>
46 #include <cstring.h>
47 #include <functional.h>
48 #include <limits.h>
49 #include <memory.h> // only to support hash of smart pointers
50 #include <stdexcept.h>
51 #include <string.h>
52 #include <type_traits.h>
53 #include <utility.h>
54 #if __cplusplus >= 201703L
55 #    include <string_view.h>
56 #endif
57
58 // #define ROBIN_HOOD_LOG_ENABLED
59 #ifdef ROBIN_HOOD_LOG_ENABLED
60 #    include <iostream.h>
61 #    define ROBIN_HOOD_LOG(...) \
62         std::cout << __FUNCTION__ << "@" << __LINE__ << ": " << __VA_ARGS__ << std::endl;
63 #else
64 #    define ROBIN_HOOD_LOG(x)
65 #endif
66
```